






1) برای ساخت برج با  $n$  آجر حتماً باید  $n-2$  آجر وسط شکل  از یک شکل طبقه  ایجاد کرد (در فیلد اول و آخر طبقه باشند) مثلاً در مثال عدد ۳ شکل حالت ۱: برج حالت 2 در این دسته قرار می گیرند. دسته دوم هم حالتی است که  بدون شکل در شکل وجود داشته باشد که در حالت اول نیازمند جواب  $2(n-2)$  هستیم. در واقع این حالت  $n-2$  را می یابیم و با اضافه کردن آن به سمت چپ و راست جواب را داریم!

همچنین حتماً با اضافه کردن  $\square$  به سمت چپ و راست جواب جدیدی را خواهیم داشت! حالت سوم هم این  $n-3$  آجر ساده می سازیم و به آن  اضافه می کنیم شکل آن به سمت راست باید دو آجر زیر دو آجر بالایی می باشد یا در واقع به شکل حالتی که  آجرها مستطی را تشکیل می دهند  $(n-4)$  از کل حالت کم کنیم پس:

$$Y(n) = Y(n-1) + Y(n-2) + Y(n-3) - Y(n-4)$$

$$\left. \begin{array}{l} Y(0) = 1 \\ Y(1) = 1 \\ Y(2) = 1 \\ Y(3) = 2 \end{array} \right\} \rightarrow \begin{array}{l} \text{مقادیر } n \\ \text{داده شده} \end{array}$$

همچنین حالت های پایه 1

حال در هر مرحله یک حلقه  $n$  یک بار را از  $(Y(4))$  شروع به برآورد می کنیم. که تا رسیدن به  $n$  بازنگار می شود. همچنین چون تمامی مراحل قبل نیز در آرایه ای ذخیره شده است تنها یک بار می توانیم به جواب برسیم پس مستند از  $O(n)$  عمل می شود!

بررسی صحت:

$$Y(4) = 1 + 1 + 2 - 1 = 3 \checkmark$$

$$Y(5) = 3 + 2 + 1 - 1 = 5 \checkmark$$

$$Y(6) = 5 + 3 + 2 - 1 = 9 \checkmark$$

۲) الگوریتم دینامیک به مسئله knapsack است

در زمان ۲۱ تابعی تعریف می‌کنیم (آریدای دو بعدی) که کمترین مقدار که استفاده شده برای هر سبد به وزن  $s$  را برمی‌گرداند  
 از جدول خود استفاده است و نه آن سبکی به حال انجام عملیات!

$s+1$

	0	1	...	$s$	...	$n$
0	0	0	...	0	...	0
1	0	1	...	1	...	1
...	...	...	...	...	...	...
$n$	0	1	...	1	...	1
$n+1$	0	1	...	1	...	1

$n \times s$   $\min(w_i)$

در درجیف بخودی چون از هر سبکی به نهایت نرسیم باید  
 اجازه ی استفاده از هر سبکی به تعداد دلخواه را دهیم  
 برای اینکه از هر سبکی به تعداد  $\frac{s}{w_i}$  در رفته‌ها  
 تکرار می‌کنیم (از آنکه به مسئله گور نیست در خوشه‌های جدید)

$$Y(s) = \begin{cases} Y(s-1) & \text{if } s \geq 1 \\ 0 & \text{if } s = 0 \\ \min\{Y(s-1) + 1, Y(s)\} & \text{if } s < 1 \end{cases}$$

مگر در این حالت به صورت row-major است پس به مسئله گور نیست زمان حل مسئله  $O(ns)$  است

ب) خیر  $O(sn)$  بهینه جدای است! اگر ورودی پهن باشد و اندازه آن در حافظه  $\log s$  است  
 یعنی اگر یک آرید به طول  $\log s$  را از  $O(s)$  حل کرده باشیم.

$$s = 2^{\log s} = s$$

پس اگر  $\log s$  را  $m$  (طول آرید فرض کنیم)  $O(s^m)$  حل شده است

که ضمیمه‌ای نسبت به واقع آنچه در کلاس به  $order$  زمانی اهمیت دارد طول درجیف از مسئله آن  
 که این چون یک عدد ثابت است و طول آن  $\log s$  است صورت این مسئله در  $order$  با یک  $order$  ثابت  
 $order$  به نسبت طول  $s$  می‌باشد



۳) الف) این مسئله را حتی با روش (bruteforce) از  $O(n^2)$  قابل حل است!

به این صورت که تمامی بازه‌ها را می‌سازیم و  $max$  آنها را برمی‌گردانیم.

```
ans = 0
for i = 1 to n
    temp = 0
    for j = i to n
        temp += A[j]
        ans = max(ans, temp)
        if temp > ans
            ans = temp
            maxi = i, maxj = j
return ans, maxi, maxj
```

موردی که این روش در مقابل آمده است!

موتور - پس ۲ حالتی تو در تو داریم

مسئله از  $O(n^2)$  حل شده است!

ب) با استفاده از رابطه بازگشتی زیر آن فرض کنیم آریه B یک آریه است که در عنصر نام آن  $max$  مجموع عناصر متوالی تا عنصر نام ذکر شده است می‌توان محاسبه  $B(n)$  حاصل جواب شده است!

$$B(i) = \max(A(i), A(i) + B(i-1))$$

محور در هر مرحله عدد ثابت. را نیز داریم برای  $B(i)$  تعیین در محاسبه  $A(i)$  مقدار  $max$  را می‌گیریم و ذخیره می‌کنیم

$m = B(1) = \max(0, A(1))$  , if  $B(1) = 0$  , start index = 0 else start index = 1  
for  $i = 2$  to  $n$  end index = 0 end index = 1

```
if A(i) > B(i-1) + A(i)
    B(i) = A(i) , s = 1
else
    B(i) = B(i-1) + A(i) , s = 2
if B(i) > m
    if (s == 1)
        end index = i , start index = i
    else (s == 2)
        end index = i
m = B(i)
```

شبهه

در واقع در هر مرحله یا خود عنصر یا مجموع هر عنصر با  $max$  مجموع زیر آریه‌ها  
- آن عنصر را بررسی می‌کنیم، چرا که  
چک می‌کنیم که آیا مجموع با اضافه شدن این عنصر  
بزرگ‌تر شود یا نه اگر نه فقط مجموع را به این  
رابطه اضافه می‌کنیم.

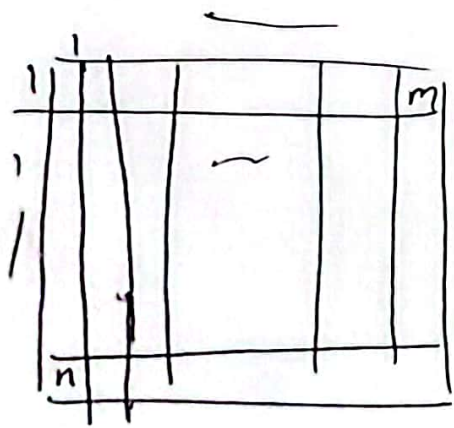
در استیلا: آرایه تجزیه یک آرایه در معنی تجزیه ایجاد می‌کنیم. این صورت که  $m(n)$  مجموع همه اعداد زیر مستطیل است که از نقطه  $(0,0)$  از مستطیل اصلی آغاز و به سمت  $(n,n)$  از مستطیل اصلی ادامه دارد.

ما داریم تمام حالت‌های جدا کردن مستطیل که به فوق از  $O(n^3)$  قابل انجام است. پس استیلا از  $O(n^3)$  مستطیل‌های قابل جدا کردن از آرایه فوق جدا می‌شوند و در یک آرایه ذخیره می‌شوند پس ما به تعداد مستطیل‌ها  $O(n)$  و بار این مستطیل‌ها  $O(n^2)$  که از  $O(n^3)$  قابل حل است!

تفاوت با سوال قبلی در این است که اینجا این می‌شود که جای حفظ شماره باید درست از مستطیل را ذخیره کنیم. (یا مستطیل که بالا، پایین، چپ، راست باشد) (یا نقطه  $(0,0)$  و  $(n,n)$  که می‌گذرد)

در واقع برای هر حالت‌های ممکن در آرایه  $m$  که تعداد ذخیره شده است، برای  $up, down$  و  $right, left$  به صورتی که  $right, left$  باشد مجموع عناصر

به ما می‌دهد و مناسب می‌شود!



(۵۱) یک جدول  $m$  در  $n$  سطر داریم. (زیر  $m$  یعنی  $\max$  کالوس دریافتی در  $n$  سطر).

$$m(i, j) = \max(m(i-1, j), m(i, j-1)) + 1$$

بالاترین حالت

$$m(i, j) = 0 \text{ if } i=0 \text{ or } j=0$$

$$m(i, 0) = 0 \text{ and } m(0, j) = 0$$

$$m(i, 0) = 0 \text{ and } m(0, j) = 0$$

جدول  $m \times n$  داده ها را به صورت row-major می بینیم پس حل مسئله در  $O(mn)$  انجام می شود

در واقع این الگوریتم با یک جدول  $m \times n$  هر خانه به بهترین حالت ممکن، جویب نمی آید و در هر خانه آخر یعنی  $m(m, n)$  به حالت  $m$  و  $n$  در واقع هر بار بهترین حالت را بین عدد  $m$  و  $n$  (زیر  $m$  و  $n$ ) در هر خانه  $m$  و  $n$  یافت می شود و با جمع کردن این بهترین حالت را در هر خانه جدول  $m \times n$  ذخیره می شود.

مسئله به شکل کلی:

```

for i = 1 to n
    m(i, 0) = 0
    for j = 1 to m
        m(i, j) = max(m(i-1, j), m(i, j-1)) + 1
    end for
end for

for j = 1 to n
    for i = 1 to m
        m(i, j) = max(m(i-1, j), m(i, j-1)) + 1
    end for
end for

return m(m, n)
    
```

فرم حل این الگوریتم در  $O(mn)$  است



(ز،نا) m را بیشترین درس ممکن که با نا سال و سبب نهند و می توان برداشت را در خود ذخیره می کند .  
و منبع سبب در قاری m(N,S) هست !

$$m(i, j) = \begin{cases} 0 & m=0 \leq n=0 \\ \max \left( m(i-1, j) - \min(c_m) + 1, m(i-1, j-1) + K_{ij} \right) & \text{در اینجا} \\ & \text{تعداد در سال نا} \end{cases}$$

در واقع جواب را با سببی هر ۳ حالت است پس سبب که شامل ① یک درس به سال تحلی نام با کمترین هزینه است  
② برداشتن هیچ درس در آن سال ③ برداشتن تمام درس در آن سال که ما جواب را باید با یک سال قبلی  
max این ۳ حالت را میگیریم .

همچنین این سبب با توجه به ابعاد جدول دانسته به صورت row-major نیز میجوید بر برگردان آن هستیم از

$$O(\max(NS, K_{max} \cdot N))$$

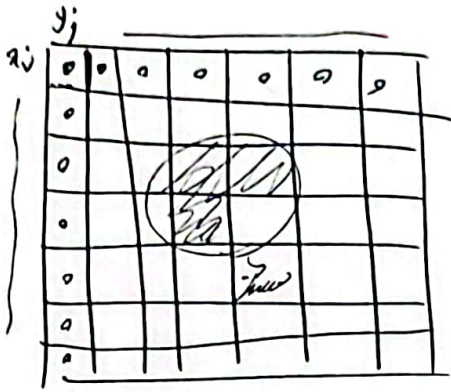
در در زمانی رو برداشت !

در این حالت سبب



چون که پس برداشتن ما از  $O(K_{max} \cdot N)$  است و برگردان جدول از  $O(NS)$  .  
برگشت برگردان جدول

اگر  $m(i, j)$  طولانی ترین زیر رشته از  $x[1..i]$  و  $y[1..j]$  است که به هم می رسد.



$$m(i, j) = \begin{cases} m(i-1, j-1) + 1 & x[i] = y[j] \\ \max(m(i, j-1), m(i-1, j)) & x[i] \neq y[j] \end{cases}$$

$$i=0 \text{ یا } j=0 \rightarrow m(i, j) = 0$$

در واقع دو حالت اصلی وجود دارد:

1)  $x_m = y_n$ : any sequence  $z'$  that does not ends in  $x_m y_n$

can be made longer by adding  $x_m y_n$  to the end

hence LCS  $z$  must ends in  $x_m y_n$

2)  $x_m \neq y_n$  and  $z_x \neq x_m$

چون  $z$  -  $x_m$  قسم نمی شود پس  $z$  یکی از رشته های  $x[1..m-1]$  و  $y[1..n]$  است

و به همین دلیل  $x_m \neq y_n$  و  $z_x \neq x_m$  است.

LCS\_Length( $x, y$ ) {

$m, n = \text{length}(x), \text{length}(y)$

$c[i, i=1 \text{ to } m, 0] = 0$

$c[0, j, j=1 \text{ to } n] = 0$

for  $i=1$  to  $m$

for  $j=1$  to  $n$

if  $x[i] = y[j]$

$c[i, j] = c[i-1, j-1] + 1$

$b[i, j] = \text{"\u2191"}$

else if  $c[i-1, j] > c[i, j-1]$

$c[i, j] = c[i-1, j]$

$b[i, j] = \text{"\u2191"}$

else

$c[i, j] = c[i, j-1]$

$b[i, j] = \text{"\u2190"}$

return  $c, b$

● الگوریتم LCS از اردو زمانی  $O(mn)$  به بخش الف حل شد

ب. اعلان به بخش ب باید این رشته مشترک را با استفاده از آرایه b که در روش قبل ضمیمه برینت کنیم  
~ " یعنی یک عنصر مشترک دیده ایم."

```
LCS-print(b, x, i, j) {  
    if (! (i & j))  
        return  
    if b(i, j) == ~  
        LCS-print(b, x, i-1, j-1)  
        print x[i]  
    else if b(i, j) == ↑  
        LCS-print(b, x, i i-1, j)  
    else  
        LCS-print(b, x, i, j-1)  
}
```

این الگوریتم نیز از  $O(m+n)$  استفاده می کند. با در واقع  $O(\max(m, n))$



این سوال همان سوال Corporate party plan هست.

(۸)

بوی اصل این مسئله  $A(i)$  را بهترین مکانی در صندلی که  $i$  رئیس باشد و فقط  $i$  می‌نشیند

$$A(i) = \begin{cases} u_i + \sum_{j \in \text{grandchild}(i)} A(j) & i \text{ is invited} \\ \sum_{j \in \text{child}(i)} A(j) & i \text{ isn't invited} \end{cases}$$

بهترین  $B(i)$  را بهترین مکانی در صندلی که  $i$  رئیس باشد و همورس  $i$  می‌نشیند و همورس  $i$  می‌نشیند

$$B(i) = \sum_{j \in \text{child}(i)} A(j)$$

$$A(i) = \max \left\{ u_i + \sum_{j \in \text{child}(i)} B(j), B(i) \right\}$$

سین در نهایت

Party planning( $u, i$ )

شبهه

visited( $i$ ) = True

$A(i) = u_i$

$B(i) = 0$

for  $j \in i$ 's children

party planning( $u, j$ ) // DFS

$A(i) += B(j)$

$B(i) += A(j)$

return ( $\max(A(i), B(i))$ )

بهینه‌ترین زمان اجرای این برنامه از  $O(n)$  و پیچیدگی حافظه نیز از  $O(n)$  است.

(9) لم از استقر، استقرا کنیم تا رابطه را ثابت کنیم.  $\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$

- پایه استقر،  $n=1$

$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \checkmark$$

- فرض استقر،  $n=k$  فرض ما شد  $\begin{bmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^k$

$$\text{حکم:} \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^k \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F_{k+1} & F_k \\ F_k & F_{k-1} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} F_{k+1} + F_k & F_{k+1} \\ F_k + F_{k-1} & F_k \end{bmatrix} = \begin{bmatrix} F_{k+2} & F_{k+1} \\ F_{k+1} & F_k \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{k+1}$$

ثابت شد! حال می‌دانیم از لم فوق برای اثبات مرحله بعدی استفاده کنیم.

اضافه) در غیرنهایی می‌دانیم رابطه را بنویسیم.

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} = \dots = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n/2}$$

مغایرت فوق را می‌دانیم به صورت یک دقت فرض کنیم حال با استفاده از  $memoization$  می‌توانیم هر بار که یک مدیسین در آن اتفاق می‌افتد مقدار آن را ذخیره می‌کنیم تا از محاسبه مجدد آن بزرگتر شود و می‌دانیم که غیرنهایی

$$F_n = F_{n-1} + F_{n-2}$$

سپس در رسم دقت برای محاسبه غیرنهایی تنها یک مسیر از ریشه تا برگ می‌توانیم رسم کنیم و در این

مسیر تمام نودانها برای محاسبه محاسبه شده است و مقدار ذخیره شده است

سپس با سبب هم اندیشه ارتفاع دقت کافی است. ارتفاع دقت با  $n$  برابر هم  $\log n$  است

پس این مقدار هم  $O(\log n)$  (کلام می‌شود)



$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix} = \begin{bmatrix} f_{n+1} \\ f_n \end{bmatrix} \quad \text{در واقع چون می دانیم } (f_{n+1}, f_n, f_{n-1})$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f_{n-1} \\ f_{n-2} \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix}$$

آنها این عبارت را در عبارت اول جایگزین کنیم توانیم به حالت صفی قبل می رسیدیم بدست می آید تا جایی که توان  $n$  از  $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$  بدست می آید و توانی  $\begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$  که معلوم است با  $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$  ضرب می شود که این حالت خاص و بدست می آید

پس در واقع برای ماسی  $\begin{bmatrix} f_n \\ f_{n-1} \end{bmatrix}$  به یک حالت برای ماسی  $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$  که می شود که حال طور که بیان شد در  $O(\log n)$  است.

(ب) حال برای ماسی مستقیم به ماسی نزدیکتر مشابه با همان ماسی  $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$  طراحی کنیم که در ضرب کردن آن به ماسی  $\begin{bmatrix} f_{n-1} \\ f_{n-2} \end{bmatrix}$  یک ماسی بینان خوب از  $f_n$  تا  $f_{n-k+1}$  را پیدا کنیم. در واقع چون این حالت تقسیم لا فکتی حالت قبل است و یعنی حالت قبل همین  $k-2$  بود و است ماسی

$$f_n, f_{n-1} + f_{n-2} + \dots + f_{n-k}$$

همین این بار هر ماسی در وقت به جای دو نصف شدن  $k$  نفس می خورد پس به جای اینکه از  $O(\log n)$  مستقیم شود از  $O(\log n / k)$  می شود حل می شود و همینال کار می خواهد باشد



روش کلی برای این مسئله بدین روش است:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_{n-1} \\ f_{n-2} \\ \vdots \\ f_{n-k} \end{bmatrix} = \begin{bmatrix} f_n \\ f_{n-1} \\ \vdots \\ f_{n-k+1} \end{bmatrix}$$

$K \times K$ 
 $K \times 1$

بدین روش در هر مرحله اول جمع‌های متوالی را می‌سازیم و یک‌های تنها در درجه‌های بعدی  
 هر یک از آن در درجه‌ها را می‌بندیم و مجموع‌های  $f_{n-k+1}$  تا  $f_n$  را می‌سازیم تا  $k$  تا  $k+1$   
 که می‌شود از  $f_{n-1}$  تا  $f_{n-k+1}$  که با هم جمع می‌شوند تا  $f_n$  و به این  
 در واقع هر یک از یک فرزند از این روشی است و بدین

در نهایت  $O(\log_2 n)$  (کمترین زمان)

کمترین  $O(\log_k n)$